連載

# FPGAで学ぶVerilog HDL



第4回（最終回）

## ディジタル時計を作る

デバイスの記事　ビギナーズ　関連データ

井倉将実

**前回（2005年11月号，pp.117-122）は，180秒のタイマを設計しました．180秒タイマは，3けたのカウンタを順次回すことでカウント・ダウンするものでした．今回はこれを機能アップし，ディジタル時計を作ります．なお，本稿で紹介する設計データは，本誌Webサイト（http://www.cqpub.co.jp/dwm/）からダウンロードすることができます．（筆者）**

今回は，電源投入直後（またはFPGA起動直後）から時を刻み，時刻を表示する時計を設計します．普通の時計では，時刻を設定する方法が用意されています．しかし，時刻設定の機能はかなり複雑な処理になるため，今回は実装しません．

## 1．ディジタル時計の仕様を考える

本連載で使用するFPGAボード(1)には，3けたの7セグメントLEDが搭載されています．しかし，3けたでは，「12時34分56秒」と表示できません．6けたもある時刻の情報を少ないけた数で表示するため，今回は以下のような方法をとります．

### ● 分割表示を行う

「時：分」と「秒」を切り替えて表示します．切り替えには，FPGAボード上のスイッチSW3を使用します．

スイッチを押していない状態（電気的には“H”レベル）では，7セグメントLED上に「時：分」を表示します．スイッチを押している状態（電気的には“L”レベル）であれば「秒」を表示します．

「時：分」の表示は，「11：59」のように4けた必要です．これを3けたで表現するために，時のけたは16進表示にします．つまり，1時から9時まではそのまま数字を表示し，10時のときは「A」，11時のときは「B」とします．例えば，11時59分であれば「B 5 9」と表示します．また，1分のけたの7セグメントLEDに用意されている「.（ドット・ポイント）」を1秒ごとに点滅させます．

「秒」の表示では，3けたのうちの下2けたに秒の値を表示します．3けた目は「－」を表示します．

### ● 時を早く刻む

ディジタル時計は，0時00分00秒から始まり，11時59分59秒まで進んだら次は再び0時00分00秒に戻ります（12時間表示の場合）．実際に時計として使っているときに，これで不つごうを感じることはありません．

しかし，設計した回路の動作を見たい（検証したい）ときはどうでしょうか．1時間に1回しか起こらない「時」のけたのカウントアップや，12時間に1回しか起こらない0時00分00秒に戻る動作を待ってはいられません．

そこで，1秒の単位を加速する機能を加えることにします．例えば，本来は1秒単位で駆動する秒タイミング信号を1/30秒で駆動するようにすることで，30倍早く時を刻む時計になります．加速する動作を「ブースト・モード」と名づけます．スイッチSW4を押している間だけ時間を30倍早く進ませることにします．これで，1分が2秒で，1時間が2分で進むことになります．こうすることで，けた上がりの状態を目で観測しやすくなります．

実は，ブースト・モードを利用すると，時刻合わせもできます．実際の時計のように使いやすい時刻セットの機能は，みなさん自身の知恵で実現していただきたいと思います．秒カウンタのリセット機能と分カウンタのカウントアップ機能さえあれば，それなりに使える時計になります．

## 2. 回路設計のポイント

ディジタル時計のVerilog HDLソース・コードを**リスト1**に示します．今回は，新しい構文は一つもありません．前回までの知識の範囲で，ディジタル時計を作ることができます．

### ● 時，分，秒のけたの回路――時を刻む動作

あたりまえのことですが，ディジタル時計は，時，分，秒を数えます．

回路の主要部分は180秒タイマの応用です．大きな違いは，180秒タイマではカウント・ダウンしていましたが，ディジタル時計ではカウント・アップするところです．また，「59秒」や「59分」の次に上のけたをカウント・アップするように，条件判断の動作が必要です．

時，分，秒の各けたは，1秒ごとに動作するカウンタを実装します．また，分と秒は，それぞれ2けたのカウンタにします．すべてのけたカウンタは，直前までのカウンタ値を1秒ごとに評価して動作を決めます．

これらの動作を細かく見てみると，以下のような制御を行うことで，時計として機能します．

- 1秒のけたカウンタはフリー・ランで動く．ただし，自身が9であれば0に戻る．
- 10秒のけたカウンタは，1秒のけたカウンタが9であれば＋1する．ただし，自身が5であれば0に戻る．
- 1分のけたカウンタは，10秒のカウンタが5，1秒のカウンタが9(59秒)であれば＋1する．ただし，自身が9であれば0に戻る．
- 10分のけたカウンタは，1分，10秒，1秒のカウンタが順に9，5，9(9分59秒)であれば＋1する．ただし，自身が5であれば0に戻る．
- 時のけたカウンタは，10分，1分，10秒，1秒のカウンタが順に5，9，5，9(59分59秒)であれば＋1する．ただし，自身が11であれば0に戻る．

このように，自分のカウンタの値をどのように進めるべきかを決めるため，1秒ごとにほかのカウンタの値を確認することがポイントになります．

### ● 7セグメントLED表示部――16進数表示への拡張

前回の180秒タイマでは，0～9の数字以外のコードを判別したら「－」(マイナス)を表示させていました．今回の時計の場合，「分」と「秒」はまったく同じ機能を使えます．しかし，「時」の部分は10時のときにAを，11時のときにBを表示させなければなりません．そこで，16進数を表示する回路への拡張が必要です．

そこで今回は，%1010と%1011に，それぞれAとBを表す7セグメントLEDの表示パターンを追加しました．

### ● 1秒タイミング生成回路

1秒のタイミングを生成する機能は，基本的に前回の180秒タイマと変わりません．ただし，今回はブースト・モードを追加しています．

スイッチの値を読み取ることで，ブースト・モードであるかどうかを判定します．内部に用意されているフリー・ランニングの一つのアップ・カウンタの値を，モードごとに異なる判別方法で利用します．

ブースト・モード時は，スイッチの値は電気的に“L”レベルになります．よって，まず`if ( SW_VMAX == 1'b0)`で，スイッチが押されているかどうかを判断し，スイッチが押されているときは1/30秒のパラメータ値になったらPulse_1sec信号をセットします．スイッチが押されていないときは`else`以降の条件文に移行し，1秒のパラメータ値で判断します．上段と下段では条件文に使用するカウンタ比較パラメータが違います．

このようにすることで，1秒単位と1/30秒単位を切り替えてPulse_1sec信号を出力できます．

### ● 動作確認

米国Altera社のQuartus IIを使って，論理合成と配置配線を行います．ピン配置は**表1**のようになります．

でき上がった回路データをFPGAにダウンロードすると，7セグメントLEDには「000」が表示されるはずです．そして，いちばん右のけたにある「.」が1秒ごとに点滅を繰り返します．

SW3を押すと，7セグメントLED上には「－」と2けたの数字が現れ，数字は1秒ごとにアップカウントします．「59」の次には「00」に戻ります．

SW4を押します．30倍早く時間が進みます．分や時の単位がどんどん加算されていくのがわかります．同時にSW1を押して秒表示にすると，ほとんど見えないくらい高速にカウント・アップしています．

これでディジタル時計の動作を確認できました.

## 3. 連載のまとめ

前回は180秒タイマ，今回はディジタル時計と，2種類のアプリケーションを例に，Verilog HDLを解説しました.

複数のparameter文を条件判定に使ったり，7セグメント表示回路でif文とcase文を使い分けたりと，一つの回路を作るにしてもさまざまなアプローチがあります.

今回のディジタル時計はひたすら動き続ける回路です．そのため，各けたのカウンタを構成する回路以外には，条件比較を行うような回路はありません.

ここで，「ある時間になったら，ある処理を行う」という機能が必要だったとしたらどうでしょうか．「10時30分になったら，ブザーを鳴らせ」というようなことも，可能なような気がしませんか？

これも結局は，時・分（必要であれば秒も）カウンタのすべてのけたをつねに監視する回路を入れておき，「もし設定された値になったら，決められた操作を行う」という構成にしておけば実現できます．回路規模さえ許せば，決まった時間に決まった動作を行う回路をいくらでも実装できます．このように，どんどん機能を拡張して，自分なりのオリジナル時計を考えてみてください.

本連載では，あえてVerilog HDLの基本中の基本と言える構文しか紹介しませんでした．Verilog HDLには，ほかにもいろいろな書きかたがあります．参考文献なども参考にして，ぜひ次のステップへと進んでください.

本連載が，Verilog HDLを習得しようと考えている読者のみなさんのお役に立てば幸いです.

**表1　ピン配置**

| ピンの名まえ | 入出力 | 接　続 | ピン番号 |
|---|---|---|---|
| SEG_3［0］ | 出力 | 7セグメントLED（D3）のdp | 21 |
| SEG_3［1］ | 出力 | 7セグメントLED（D3）のg | 16 |
| SEG_3［2］ | 出力 | 7セグメントLED（D3）のf | 15 |
| SEG_3［3］ | 出力 | 7セグメントLED（D3）のe | 20 |
| SEG_3［4］ | 出力 | 7セグメントLED（D3）のd | 22 |
| SEG_3［5］ | 出力 | 7セグメントLED（D3）のc | 19 |
| SEG_3［6］ | 出力 | 7セグメントLED（D3）のb | 14 |
| SEG_3［7］ | 出力 | 7セグメントLED（D3）のa | 13 |
| SEG_2［0］ | 出力 | 7セグメントLED（D2）のdp | 8 |
| SEG_2［1］ | 出力 | 7セグメントLED（D2）のg | 5 |
| SEG_2［2］ | 出力 | 7セグメントLED（D2）のf | 99 |
| SEG_2［3］ | 出力 | 7セグメントLED（D2）のe | 7 |
| SEG_2［4］ | 出力 | 7セグメントLED（D2）のd | 9 |
| SEG_2［5］ | 出力 | 7セグメントLED（D2）のc | 6 |
| SEG_2［6］ | 出力 | 7セグメントLED（D2）のb | 98 |
| SEG_2［7］ | 出力 | 7セグメントLED（D2）のa | 97 |
| SEG_1［0］ | 出力 | 7セグメントLED（D1）のdp | 94 |
| SEG_1［1］ | 出力 | 7セグメントLED（D1）のg | 81 |
| SEG_1［2］ | 出力 | 7セグメントLED（D1）のf | 87 |
| SEG_1［3］ | 出力 | 7セグメントLED（D1）のe | 93 |
| SEG_1［4］ | 出力 | 7セグメントLED（D1）のd | 96 |
| SEG_1［5］ | 出力 | 7セグメントLED（D1）のc | 86 |
| SEG_1［6］ | 出力 | 7セグメントLED（D1）のb | 85 |
| SEG_1［7］ | 出力 | 7セグメントLED（D1）のa | 84 |
| SW_VMAX | 入力 | スイッチ（SW4） | 31 |
| SW_DispSEC | 入力 | スイッチ（SW3） | 32 |
| SW_RESET | 入力 | スイッチ（SW2） | 33 |
| SW_START | 入力 | スイッチ（SW1） | 34 |
| SYSCLK | 入力 | クロック発振器 | 39 |

**参考・引用*文献**

(1) 山際伸一；FPGAボードで学ぶ論理回路設計，CQ出版，2003年5月.

**いくら・まさみ**
**来栖川電工（有）中野R&Dカンパニ**
**FPGAコンソーシアム会員**

**リスト1　ディジタル時計のVerilog HDLソース・コード**

```
module DigClock(
// 入力ピン
    SYSCLK,
    SW_RESET,

    SW_START,
    SW_DispSEC ,          // 秒単位表示スイッチ.
                          // このスイッチが押されていないと
                          // 時分表示. 押されると秒表示
    SW_VMAX ,             // 試験用秒カウンタの加速装置

// 出力ピン
    SEG_1 ,
    SEG_2 ,
    SEG_3
);

// 入出力ピンの定義
    input              SYSCLK ;
    input              SW_RESET ;

    input              SW_START ;
    input              SW_DispSEC ;
    input              SW_VMAX ;
```

リスト1　ディジタル時計のVerilog HDLソース・コード(つづき)

```
    output     [7:0]     SEG_1 ;
    output     [7:0]     SEG_2 ;
    output     [7:0]     SEG_3 ;

    wire       [7:0]     SEG_1 ;
    wire       [7:0]     SEG_2 ;
    wire       [7:0]     SEG_3 ;

// 内部回路用のレジスタ宣言
    reg                  CLOCK_Start ;

    parameter  [31:0]    param_1Second   = 32'h01F78A40 ;

                              // 33MHz時の1秒タイミング
    parameter  [31:0]    param_30mSecond = 32'h0010C8E0 ;

                              // 33MHz時の1/30秒タイミング

    reg        [31:0]    Sec1_counter ;

    reg        [3:0]     Count_1sec   ;
    reg        [2:0]     Count_10sec  ;
    reg        [3:0]     Count_1min   ;
    reg        [2:0]     Count_10min  ;
    reg        [3:0]     Count_hour   ;

    wire                 Cnt_1sec_is_9 ;
    wire                 Cnt_sec_is_59 ;
    wire                 Cnt_1min_is_9 ;
    wire                 Cnt_min_is_59 ;

    reg        [ 7:0]    node_SEG3 ;  // 7Seg用内部ノード
    reg        [ 7:0]    node_SEG2 ;  // 7Seg用内部ノード
    reg        [ 7:0]    node_SEG1 ;  // 7Seg用内部ノード

    reg                  Pulse_1Sec ;
    reg                  SEC_Prompt ;
    reg                  TIM_START ;
    reg                  TIMEOUT ;

/*******************************************************/
// 1秒タイミング生成
/*******************************************************/
always @( posedge SYSCLK or negedge SW_RESET )
begin
    if ( SW_RESET == 1'b0 ) begin
       Sec1_counter <= 32'b0 ;
       Pulse_1Sec   <= 1'b0 ;
       SEC_Prompt   <= 1'b0 ;
       CLOCK_Start  <= 1'b0 ;
    end else begin
        CLOCK_Start <= SW_START ;
        if ( SW_VMAX == 1'b0 ) begin
                            // ブースト・モード・スイッチの確認
            if ( Sec1_counter == param_30mSecond ) begin
                                // 1/30ごとに秒フラグのセット
                Pulse_1Sec      <= 1'b1 ;
                Sec1_counter    <= 32'b0 ;
            end else begin
                Pulse_1Sec      <= 1'b0 ;
                Sec1_counter    <= Sec1_counter + 1 ;
            end
        end else begin
            if ( Sec1_counter == param_1Second ) begin
                                 // 1秒ごとに秒フラグのセット
                Pulse_1Sec      <= 1'b1 ;
                Sec1_counter    <= 32'b0 ;
            end else begin
                Pulse_1Sec      <= 1'b0 ;
                Sec1_counter    <= Sec1_counter + 1 ;
            end
        end

        if ( Pulse_1Sec == 1'b1 ) begin
            SEC_Prompt   <= ~SEC_Prompt ;
                             // 秒単位のプロンプト表示レジスタ
        end
    end
end
```

```
/*******************************************************/
// 秒の位カウンタ回路
/*******************************************************/
always @( posedge SYSCLK or negedge SW_RESET )
begin
    if ( SW_RESET == 1'b0 ) begin
        Count_1sec  <= 4'b0 ;     // Set 0Second
        Count_10sec <= 3'b0 ;     // Set 0Second
    end else begin
        if ( CLOCK_Start == 1'b1 & Pulse_1Sec == 1'b1 )
                               begin// 時計スタートになっていれば
        // 1秒の位チェック
            if ( Cnt_1sec_is_9 == 1'b1 ) begin
                                    // 9秒になったかをチェック
                Count_1sec <= 4'b0 ;
            end else begin
                Count_1sec <= Count_1sec + 1 ;
            end
        // 10秒の位チェック
            if ( Cnt_sec_is_59 == 1'b1 ) begin
                                   // 59秒になったかをチェック
                Count_10sec <= 3'b0 ;
            end else begin
                if ( Cnt_1sec_is_9 == 1'b1 ) begin
                    Count_10sec <= Count_10sec + 1 ;
                end
            end
        end
    end
end

    assign     Cnt_1sec_is_9 = ( Count_1sec ==
     4'b1001                         ) ? 1'b1 : 1'b0 ;
                                   //  9秒になったかをチェック
    assign     Cnt_sec_is_59 = ( Count_1sec ==
    4'b1001 & Count_10sec == 3'b101 ) ? 1'b1 : 1'b0 ;
                                   // 59秒になったかをチェック

/*******************************************************/
// 分の位カウンタ回路
/*******************************************************/
always @( posedge SYSCLK or negedge SW_RESET )
begin
    if ( SW_RESET == 1'b0 ) begin
        Count_1min  <= 6'b0 ;     // Set 0Minute
        Count_10min <= 4'b0 ;     // Set 0Minute
    end else begin
        if ( CLOCK_Start == 1'b1 & Pulse_1Sec == 1'b1 )
                          begin    // 時計スタートになっていれば
            if ( Cnt_sec_is_59 == 1'b1 ) begin
            // 1分の位チェック
                if ( Cnt_1min_is_9 == 1'b1) begin
                                    // 9分になったかをチェック
                    Count_1min  <= 6'b0 ;// Set 0Minute
                end else begin
                    Count_1min    <= Count_1min + 1 ;
                end
            // 10分の位チェック
                if ( Cnt_min_is_59 == 1'b1 ) begin
                                  // 59分になったかをチェック
                    Count_10min <= 3'b0 ;
                end else begin
                    if ( Cnt_1min_is_9 == 1'b1) begin
                                  // 9分になったかをチェック
                        Count_10min <= Count_10min + 1 ;
                    end
                end
            end
        end
    end
end

    assign    Cnt_1min_is_9 = ( Count_1min ==
     4'b1001                         ) ? 1'b1 : 1'b0 ;
                                   //  9分になったかをチェック
    assign    Cnt_min_is_59 = ( Count_1min ==
    4'b1001 & Count_10min == 3'b101 ) ? 1'b1 : 1'b0 ;
                                   // 59分になったかをチェック
```

```
/*****************************************************/
// 時の位カウンタ回路
/*****************************************************/
always @( posedge SYSCLK or negedge SW_RESET )
begin
    if ( SW_RESET == 1'b0 ) begin
        Count_hour       <= 4'b0 ;          // Set 0hour
    end else begin
        if ( CLOCK_Start == 1'b1 & Pulse_1Sec == 1'b1 )
                        begin  // 時計スタートになっていれば
            if ( Cnt_min_is_59 == 1'b1 & Cnt_sec_is_59
                        == 1'b1 ) begin  // 59分59秒か？
            if ( Count_hour == 4'b1011 ) begin
                             // 0xB=11時になったかをチェック
                Count_hour       <= 4'b0 ; // Set 0hour
            end else begin
                Count_hour       <= Count_hour + 1'b1 ;
                                              // Set 1hour
                end
            end
        end
    end
end

/*****************************************************/
// 7セグメントLED表示部
/*****************************************************/
always @( posedge SYSCLK )
begin
    if ( SW_DispSEC == 1'b1 ) begin
        node_SEG3   <= 8'b01000000 ;    // '-' display

        case ( Count_10sec )
            3'b000  : node_SEG2 <= 8'b00111111 ;
                                          // '0' display
            3'b001  : node_SEG2 <= 8'b00000110 ;
                                          // '1' display
            3'b010  : node_SEG2 <= 8'b01011011 ;
                                          // '2' display
            3'b011  : node_SEG2 <= 8'b01001111 ;
                                          // '3' display
            3'b100  : node_SEG2 <= 8'b01100110 ;
                                          // '4' display
            3'b101  : node_SEG2 <= 8'b01101101 ;
                                          // '5' display
            default : node_SEG2 <= 8'b01000000 ;
                                          // '-' display
        endcase

        case ( Count_1sec )
            4'b0000 : node_SEG1 <= 8'b00111111 ;
                                          // '0' display
            4'b0001 : node_SEG1 <= 8'b00000110 ;
                                          // '1' display
            4'b0010 : node_SEG1 <= 8'b01011011 ;
                                          // '2' display
            4'b0011 : node_SEG1 <= 8'b01001111 ;
                                          // '3' display
            4'b0100 : node_SEG1 <= 8'b01100110 ;
                                          // '4' display
            4'b0101 : node_SEG1 <= 8'b01101101 ;
                                          // '5' display
            4'b0110 : node_SEG1 <= 8'b01111101 ;
                                          // '6' display
            4'b0111 : node_SEG1 <= 8'b00100111 ;
                                          // '7' display
            4'b1000 : node_SEG1 <= 8'b01111111 ;
                                          // '8' display
            4'b1001 : node_SEG1 <= 8'b01101111 ;
                                          // '9' display
            default      : node_SEG1 <= 8'b01000000 ;
                                          // '-' display
        endcase

    end else begin

        case ( Count_hour )
            4'b0000 : node_SEG3 <= 8'b00111111 ;
                                          // '0' display
            4'b0001 : node_SEG3 <= 8'b00000110 ;
                                          // '1' display
            4'b0010 : node_SEG3 <= 8'b01011011 ;
                                          // '2' display
            4'b0011 : node_SEG3 <= 8'b01001111 ;
                                          // '3' display
            4'b0100 : node_SEG3 <= 8'b01100110 ;
                                          // '4' display
            4'b0101 : node_SEG3 <= 8'b01101101 ;
                                          // '5' display
            4'b0110 : node_SEG3 <= 8'b01111101 ;
                                          // '6' display
            4'b0111 : node_SEG3 <= 8'b00100111 ;
                                          // '7' display
            4'b1000 : node_SEG3 <= 8'b01111111 ;
                                          // '8' display
            4'b1001 : node_SEG3 <= 8'b01101111 ;
                                          // '9' display
            4'b1010 : node_SEG3 <= 8'b01110111 ;
                                          // 'A' display
            4'b1011 : node_SEG3 <= 8'b01111100 ;
                                          // 'b' display
            default : node_SEG3 <= 8'b01000000 ;
                                          // '-' display
        endcase

        case ( Count_10min )
            3'b000  : node_SEG2 <= 8'b00111111 ;
                                          // '0' display
            3'b001  : node_SEG2 <= 8'b00000110 ;
                                          // '1' display
            3'b010  : node_SEG2 <= 8'b01011011 ;
                                          // '2' display
            3'b011  : node_SEG2 <= 8'b01001111 ;
                                          // '3' display
            3'b100  : node_SEG2 <= 8'b01100110 ;
                                          // '4' display
            3'b101  : node_SEG2 <= 8'b01101101 ;
                                          // '5' display
            default : node_SEG2 <= 8'b01000000 ;
                                          // '-' display
        endcase

        case ( Count_1min )
            4'b0000 : node_SEG1 <= 8'b00111111 ;
                                          // '0' display
            4'b0001 : node_SEG1 <= 8'b00000110 ;
                                          // '1' display
            4'b0010 : node_SEG1 <= 8'b01011011 ;
                                          // '2' display
            4'b0011 : node_SEG1 <= 8'b01001111 ;
                                          // '3' display
            4'b0100 : node_SEG1 <= 8'b01100110 ;
                                          // '4' display
            4'b0101 : node_SEG1 <= 8'b01101101 ;
                                          // '5' display
            4'b0110 : node_SEG1 <= 8'b01111101 ;
                                          // '6' display
            4'b0111 : node_SEG1 <= 8'b00100111 ;
                                          // '7' display
            4'b1000 : node_SEG1 <= 8'b01111111 ;
                                          // '8' display
            4'b1001 : node_SEG1 <= 8'b01101111 ;
                                          // '9' display
        default: node_SEG1 <= 8'b01000000 ;
                                          // '-' display
        endcase
    end
end

/*****************************************************/
    assign    SEG_3[7:0]  = ~node_SEG3 ;
    assign    SEG_2[7:0]  = ~node_SEG2 ;
    assign    SEG_1[7:0]  = ( SW_DispSEC == 1'b1 ) ?
                    { SEC_Prompt, ~node_SEG1[6:0] } :
                                    ~node_SEG1[7:0] ;

endmodule
```